In the Specification:

Please include the following abstract of the disclosure on page 17 of the specification:

Abstract of the Disclosure

There is disclosed ~~disclose~~ apparatus and a method of de-fragmenting file allocations on a disk comprising: determining what pages should be swapped among the various allocations made by the operating system (OS), the OS's file system mapping updated to reflect the swapped pages and a history of the original state prior to any update recorded by the engine, the swaps performed by manipulation of the engine's data structures and/or actually exchanging data on disk where OS visible data is read and written but the original state of each altered page is not directly recorded in the historic log, but instead, a record is additionally logged of the locations of the swapped data so that an image of the OS visible data can be reconstructed prior to time of the de-fragmentation by knowing what data to effectively re-swap and what OS mapping data to effectively restore. The apparatus and method may include the step of incorporating desired close proximity information of various OS visible pages into the algorithm executed by the engine that determines what is actually swapped, in order to reasonable maintain physical close proximity of data allocated by the OS but physically re-mapped by the engine.

Please replace the paragraphs beginning at page 6, line 14 and ending at page 9, line 9 with the following paragraphs:

To gain further insight into the present invention, Figure 3 walks through a traditional de-fragmentation process. Figure 3a illustrates the initial state of the system. The disk contents are shown on the left side, with each page 301 identified 0 through 12 (i.e., the disk consists of 13 pages). The pages are divided into two sets, one group corresponding to those 303 visible by the OS and the other group reserved for tracking changes (history buffer 305). The roles of these pages are somewhat fluid as they go from holding data associated with the current main image (303, data visible to the OS) to

holding the original (historic, 305) states of pages overwritten by the OS. The various methods for managing the pages are discussed in U.S. Patent No. 6,016,553. The purpose of this figure is to illustrate the use of disk pages for main 303 and historic 305 data.

The top group of pages 0 through 7 start out holding main image data 303. The bottom group of pages 8 through 12 hold historic data 305, with the associated location field off to the right. This field, along with maps and other internal information, are maintained on disk by the engine in pages not shown in this example. The "next write" pointer 307 indicates the next location to recycle (oldest historic data). As new data is written by the OS, it is diverted to the "next write" location in the bottom pages 305 and the associated location field is updated with the original OS write's disk location. In the center of the figure is the main map through which the engine translates OS disk read requests to the actual (physical) disk location containing the desired data. At right are shown the current contents 311 of the disk as viewed by the OS through the main map 309. Since the main map 309 is initially empty, reflecting a "no mapping state," the current image visible to the OS, locations 0 through 7, directly correspond to the contents of these actual disk locations 303.

Initially the disk contains pages making up two files A and B. The allocation of file A is fragmented in that an allocation for file B is intermixed among the pages allocated to A. Disk location #4 holds the OS's directory 313, which is detailed off to the right. The directory 313 indicates the locations associated with each file. The pages containing data t1, t2, and t3 correspond to "unused" storage by the OS (free disk space 315). Pages 8 through 12 initially contain historic data 305. It is the historic data 305 that one does not want to lose, as much as possible, during a de-fragmentation.

Figure 3b illustrates the first step of a typical de-fragmentation utility. It decides to exchange the contents of pages 0 and 1 so that all the allocations associated with file A are next to each other, thus transforming the sequence "Al B1 A2 A3" into "B1 Al A2 A3." The directory 313 is also updated to reflect the new assignments. As the de-fragmentation utility intends to overwrite the data in pages 0 and 1, it must get this data out of the way. The first step is to copy the data Al and B1 to unused locations t1 and t2. Let us presume the de-fragmentation utility is unaware of the recovery methods of U.S.

Patent No. 6,016,553. Thus, what has really happened when the data Al and B1 was overwritten on top of tl and t2, is that the writes were diverted by the engine and the main map 309 updated.

Figure 3c shows the results of the de-fragmentation utility's moving Al and B l into place and updating the directory. Now as the OS views the disk through the main map 309 the disk appears de-fragmented. Generally, various background disk re-arranging activity of the engine re-optimizes the placement of data on disk to take advantage of the de-fragmentation (i.e., so that file A's allocations are near other). What is important to notice in Figure 3c is that all of the original historic data 305 H1 through H5 has been discarded in order to save a record of the modified locations. Because all the original states have been preserved prior to the de-fragmentation, it is possible to exactly reconstruct the before de-fragmentation disk image, as viewed by the OS.

Note that it is not the intent of this example to show the specific steps and methods employed by a disk de-fragmentation utility, but only the general process and effects.

The present invention, however, provides for the same before and after tracking but without requiring so much of the history buffer to record the steps to reverse the process. Thus, if you return to the state in Figure 3a and use a new de-fragmentation utility that is aware and interacts with the engine, you get Figure 3d. It shows that the engine aware de-fragmentation utility requested the engine to swap 317 locations 0 and 1 and then it updated the directory 313. Notice that more than half of the history buffer's original contents are still present, thus allowing the user to reconstruct states further back in time as compared to Figure 3c.

The "swap" is represented in Figure 3d as a special type of entry 317 in the table of disk locations and associated location fields. The actual method of storing the swap information is likely to use some combination of the location fields stored in the history buffer headers (see U.S. Patent No. 6,016,553) and other disk data structures (such as, but not limited to, the General Logged Data pages as also described in U.S. Patent No. 6,016,553).

Figure 3e illustrates a simulated disk map 319 used to access the disk image before the de-fragmentation. The simulated map 319 is constructed by reversing the

steps recorded in the history buffer: the diversion of accesses to location #4 is ended, which restores the OS's original directory mapping 313. The swap is "undone" by performing the swap again, only this time using the links 321 in the simulated map to "exchange" the pages.

The other major operation implemented by the engine is the disk revert. A disk reversion is performed by effectively writing the changes recorded in the history buffer 305 back to the main image 303. When a note about a swap 317 is encountered while walking back toward an earlier time in the history buffer 305, the swap is simply done again to undo its original effect. Note the swap may be implemented by actually exchanging pages and/or through the use of map pointers.